TRS Implementation of the DAQ Offline Interface.

The Offline interface to DAQ is specified in the StDaqLib documentation provided by Jeff Landgraf and Micheal LeVine (`$STAR/StRoot/StDaqLib/doc/OfflineInterface.pdf`). This provides a mechanism to access TPC data in a uniform manner independent of the data format utilized. Similar interface mechanisms are envisioned for the other detectors. We discuss here the specific way to access the simulated events processed through the Tpc Response Simulator (TRS).

Defining the event

There is one main difference between DAQ and the TRS implementation. The way to define the event in the case of the simulation is done through StTrsMaker itself, so there is no need for an *EventReader* object. StTrsMaker loads the event in one of two ways:

- Full simulation of an input GEANT data file.
- Reading of a .trs file from a previous simulation.

The switching to handle either case is done at the macro level. This is done by instantiating the appropriate makers in the chain and defining their behavior using the methods of those makers. The two cases are discussed below.

In the first case, the chain that is set up in the macro would need to have an St_geant_Maker and the appropriate input .fz/.fzd file should be used:

```
>  St_geant_Maker* geant = new St_geant_Maker(“geant”);
>  geant->SetNwGEANT(10 000 000);
>  geant->SetDebug();
>  geant->SetIwtype(1);
>  Tstring InFile("myInputFile.fzd");
>  geant->SetInputFile(InFile.Data());
>  chain->SetInput("geom","geant:geom");
```

Then the StTrsMaker is instantiated:

```
>  StTrsMaker* trsMk = new StTrsMaker(“Trs”);
```

To write the TRS output to a file, one invokes a method of the maker with the file name and the number of events as argument (the types taken by the constructor are `char*` and `int` respectively):

```
>  trsMk->writeFile("test.trs", Nevents);
```

The other makers that will use StTrsMaker are instantiated after this. The chain member functions Init(), Make(), etc. are then invoked.

For the second case, the only thing needed is the StTrsMaker, and the input .trs file:

```
>  StTrsMaker* trsMk = new StTrsMaker(“Trs”);
>  trsMk->readFile("test.trs");
```

followed by the user’s maker and the chain member functions.

Data File

The .trs files have a self-documenting header that can be easily read from the UNIX command prompt for convenience:

```
rcf:~/workdir/trs> head test.trs
# Tpc Response Simulator Data File
# Created: Wed Nov  3 13:20:23 1999
# version TrsDatav1.0
# Operating System: HP-UX B.10.20 A
# Number of events 1
# Geometry Information ----
# sectors 24
# rows 45
# pads 88 96 104 112 118 126 134 142 150 158 166 174 182 98 100 102 104
106 106 108 110 112 112 114 116 118 120 122 122 124 126 128 128 130 132
134 136 138 138 140 142 144 144 144 144
##
```

This way, one can obtain the following information:

- Date the file was created
- Version of the file formats. This is useful in selecting the appropriate Reader of the file.
- Operating system where the file was created
- Number of events in the file.
- Geometry Db information with which the file was created. These parameters are read in from the file so the Db does not need to be opened.
  - Number of sectors,
  - Number of rows in each sector, and
  - Number of pads in each row starting from the first row to the last.

The rest of the file is in binary format and is machine-readable only. The size of a central Venus event is ~8MB in the current version. Once the StTrsMaker is done processing through either of these cases, the event is loaded in memory. The access is provided via the interface, and this is identical to DAQ.

Readers

The data is accessed through readers. Each detector provides a specific *DetectorReader* object, which defines the type and number of readers (decoders) for a detector. In the case of TRS, only a single type is currently provided – a *ZeroSuppressedReader*.

Through the *DetectorReader*, one gets the data via the *ZeroSuppressedReader*. The member functions are defined via a base class to ensure uniformity in DAQ & Simulation chains. The *DetectorReader* will control which version of the *ZeroSuppressedReader* is created, depending on the format of the event to be processed. If the format of the data files evolve, this will become important. For TRS, the version is specified in the constructor of the *StTrsDetectorReader* object. The version can be omitted in the constructor, and by default it will take the latest one. It is important to stress that a user should only be concerned with calling the right version in the constructor, the interface is guaranteed to be the same.

The first step to access the data is to get the `St_DataSet`, and retrieve the pointer to the event data. This pointer to the event data is then used as an argument when constructing the *StTrsDetectorReader* (which inherits from *DetectorReader*). Once the *StTrsDetectorReader* is defined, the data may be accessed through the *ZeroSuppressedReader*. The other *SectorReaders*, specified in the *DetectorReader* base class, are currently not implemented for the simulation data. These can be implemented if need arises in the future. The *ZeroSuppressedReader* is called for each sector, as in DAQ, and the data is accessed through the methods of the *ZeroSuppressedReader*. In a Maker, this would look like the following:

```
#include “StTrsMaker/include/StTrsDetectorReader.hh”
#include “StTrsMaker/include/StTrsZeroSuppressedReader.hh”
Int_t StSomeMaker::Make() {
// Get the TRS Event Data Set
St_ObjectSet* trsEventDataSet = (St_ObjectSet*) GetDataSet("Event");
// Get the pointer to the raw data.
StTpcRawDataEvent* trsEvent = (StTpcRawDataEvent*)
                                        trsEventDataSet->GetObject();

// Instantiate the DetectorReader. Version will be default if not given
string version = “TrsDatav1.0”;
StTrsDetectorReader* tdr = new StTrsDetectorReader(trsEvent, version);

// Example of how one gets the TRS data with ZeroSuppressedReader
for(int isector=1; isector<=24; isector++) {
      ZeroSuppressedReader* zsr = tdr.getZeroSuppressedReader(isector);
      if(!zsr) continue;

      // Otherwise, decode it
      unsigned char* padList;
      for(int irow=1; irow<=45; irow++) {
          int numberOfPads = zsr->getPadList(irow, &padList);

          // If there are no pads, go to the next row...
          if(!numberOfPads) continue;
          for(int ipad = 0; ipad<numberOfPads; ipad++) {

            // Note that ipad is an index, NOT the pad number.
            // The pad number comes from padList[ipad]
            int nseq;
            Sequence* listOfSequences;
            zsr->getSequences(irow,
                          static_cast<int>(padList[ipad]),
                          &nseq,
                          &listOfSequences);

          // One would do the data manipulation here!
          // For this example, just print out the ADC values.
            for(int kk=0; kk<nseq; kk++) {
                for(int zz=0; zz<listOfSequences[kk].Length; zz++) {

                  cout << " " << kk
                       << " " << zz << '\t'
                       <<
static_cast<int>(*(listOfSequences[kk].FirstAdc)) << endl;

                  listOfSequences[kk].FirstAdc++;
                } // zz

            } // Loop kk
          } // loop over pads
      } // Loop over rows
    } // Loop over sectors
return kStOk; } // Make()
```

Additional Information

TRS is found in

```
$STAR/StRoot/StTrsMaker/
```

The header files are in the `include/` directory and source files in the `src/` directory.
Note that in the previous example, *StTrsDetectorReader* and *StTrsZeroSuppressedReader* are `#include`d. These in turn automatically `#include` the base class definitions for *DetectorReader* and *ZeroSuppressedReader*. The definitions for *DetectorReader*, *ZeroSuppressedReader*, and *Sequence* are located in

```
$STAR/StRoot/StDaqLib/GENERIC/EventReader.hh
```

For questions about StDaqLib please contact Jeff Landgraf or Micheal LeVine.

This is a work in progress. For questions related to the StTrsMaker *Reader* objects, you can contact
Brian Lasiuk (lasiuk@star.physics.yale.edu) or
Manuel Calderón de la Barca Sánchez (calderon@star.physics.yale.edu).